A functional framework for nonsmooth autodiff with maxpooling functions

Bruno Després

A functional framework for nonsmooth autodiff with *max-pooling* functions

Bruno Després *bruno.despres@sorbonne-universite.fr*
LJLL, Sorbonne Universite, Paris, France

Abstract

We make a comment on the recent work Boustany (2024), by showing that the Murat & Trombetti (2003) Theorem is a simple and efficient mathematical framework for nonsmooth automatic differentiation of *maxpooling* functions. In particular it gives a the chain rule formula which correctly defines the composition of Lipschitz-continuous functions which are piecewise C^1 . The formalism is applied to four basic examples, with some tests in PyTorch. A self contained proof of an important Stampacchia formula is in the appendix.

1 Introduction

In this work we make a comment on the recent work by Boustany (2024). We will show that the Murat & Trombetti (2003) Theorem gives a natural framework for nonsmooth automatic differentiation (nonsmooth autodiff). More generally we believe that the methods developed hereafter offer a comprehensive mathematical functional framework for the description of nonsmooth autodiff. Recent contributions on mathematical issues for nonsmooth autodiff are developed in Bolte & Pauwels (2021); Bertoin et al. (2021; 2023); Boustany (2024) and references therein, based on the Clarke derivative Clarke (1990). The historical remark in Bolte & Pauwels (2021) provides insights on related topics.

Notations and developments in this self contained article are kept to the minimum.

The first part introduces the notation and provides two examples which illustrate the apparent paradox discussed in this work. In a second part let us note that similar problems have been identified in the partial differential community between 1966 Stampacchia (1963) and 1990 Kinderlehrer & Stampacchia (2000), Ambrosio & Dal Maso (1990), and that the Murat & Trombetti (2003) Theorem very conveniently describes the mathematics of nonsmooth autodiff. A preliminary work Berner et al. (2019) has already highlighted the potential interest of the Murat-Trombetti Theorem for nonsmooth autodiff. However the discussion was restricted to activation functions only, so to the best of our understanding, the scope in Berner et al. (2019) was limited to fully-connected feed-forward neural network function where the main difficulty can be ruled out by means of a trivial simplification using $R'(0) = 0$ where R is the ReLU activation function. Quite surprisingly the main illustrative example in the Murat-Trombettti contribution has exactly the structure of a modern basic convolutional neural network function (CNN) Bengio et al. (2017) with a *maxpooling* function. That is why we provide a detailed and self contained proof of the Murat-Trombetti Theorem. We hope that the discussion below will contribute to popularize this approach among the Machine Learning scientific community and to show its potential for the description of many problems that intervene in nonsmooth automatic differentiation. In the third part, we will show how to describe the solution of basic problems in the context of the Murat-Trombetti Theorem. The general conclusion will be that the gradient constructed through nonsmooth autodiff in PyTorch is systematically equal to an associated gradient in the sense of Murat-Trombetti.

The author warmly thanks his colleague François Murat for his deep explanations on mathematical methods for the differentiation of nonsmooth functions. What follows is a direct application of the ideas in Murat & Trombetti (2003), with a minor adaptation of the notation.

2 Notation and examples

A fully-connected **feed-forward neural network** function $f : \mathbb{R}^m \to \mathbb{R}^n$ can be written as

$$f = f_\ell \circ S_\ell \circ f_{\ell-1} \circ S_{\ell-1} \circ \cdots \circ f_1 \circ S_1 \circ f_0 \tag{1}$$

where the parameter ℓ is identified with the number of hidden layers. The functions f_i for $i = 1, \ldots, \ell$ are affine functions with varying input and output dimensions $(a_0, a_1, \ldots, a_{\ell+1}) \in \mathbb{N}^{\ell+2}$ with $a_i > 0$, for $i = 1, \ldots, \ell$. More precisely, $f_i(x_i) = W_i x_i + b_i \in \mathbb{R}^{a_{i+1}}$ for all $x_i \in \mathbb{R}^{a_i}$. The intermediate functions S_i for $i = 1, \ldots, \ell$ are the nonlinear activation functions Sharma et al. (2017). We adopt the normalization that $0 \leq S_i'(x) \leq 1$ for almost all $x \in \mathbb{R}$. The ReLU function $S_i = R$ with

$$R(x) = \max(0, x)$$

is an extremely popular activation function used in production codes Bengio et al. (2017). The mathematical issues discussed in this work come from the fact that the ReLU function is not differentiable at $x = 0$. The point-wise non differentiability is shared with other basic functions in Neural Networks. For example *maxpooling* used in **convolutional neural networks** Bengio et al. (2017) is based on the maximum function

$$(a, b) \mapsto \max(a, b)$$

which is also not uniquely differentiable for $a = b$. In practice *maxpooling* is activated on blocks or windows of numbers on tensors of arbitrary dimensions. With adapted natural notations Pintore & Després (2024), the representation (1) holds for CNN as well by taking f_ℓ to be the identity and S_ℓ to be a *softmax* function Bengio et al. (2017). We note that all these the activation functions S_i, and by extension *maxpooling* functions and all similar functions, are Lipschitz-continuous by assumption.

Since linear functions f_i are clearly Lipschitz, then the feed-forward function (1) is Lipschitz by composition of Lipschitz functions. One has $f \in \mathrm{Lip}(\mathbb{R}^{a_0})^{a_{\ell+1}}$. Any "intermediate" step is also Lipschitz, that is

$$f_r \circ S_r \circ \cdots \circ f_1 \circ S_1 \circ f_0 \in \mathrm{Lip}(\mathbb{R}^{a_0})^{a_{r+1}} \quad \text{for } 0 \leq r \leq \ell$$

as well as $S_r \circ \cdots \circ f_1 \circ S_1 \circ f_0 \in \mathrm{Lip}(\mathbb{R}^{a_0})^{a_r}$ for $0 \leq r \leq \ell$. The Rademacher (1919) Theorem, see also Morrey Jr (2009), states that the functions f, f_r and S_r for $0 \leq r \leq \ell$ are differentiable almost everywhere (that is up to sets of zero measure). So the gradient of f, written as a matrice, is bounded $\nabla f \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_{\ell+1}, a_0}(\mathbb{R}))$. Similarly one has

$$A_r = \nabla f_r \circ S_r \circ \cdots \circ S_1 \circ f_0 \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_{r+1}, a_0}(\mathbb{R}))$$

and

$$B_r = \nabla S_r \circ \cdots \circ S_1 \circ f_0 \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_r, a_0}(\mathbb{R})).$$

Then a natural mathematical question is to give a meaning to the chain rule formula

$$\nabla f(x) = A_\ell(x) B_\ell(x) A_{\ell-1}(x) B_{\ell-1}(x) \ldots A_1(x) B_1(x) A_0(x). \tag{2}$$

For feed-forward functions, our notations imply that $A_r(x) = W_r$ is constant with respect to x, so there is no difficulty. The main difficulty is here concentrated in the matrices $B_r(x)$. Nonsmooth autodiff with respect to the weights and biases leads to similar difficulties. To illustrate the issue and the apparent paradox, we consider two examples.

2.1 First example

The first example is degenerate in a sense. We take $f_0(x) = w_0 x$ where the weight is $w_0 \in \mathbb{R}$ and $S_0(x) = R(x)$. Then $f(x) = R(w_0 x)$ and (2) becomes

$$f'(x) = R'(w_0 x) w_0, \tag{3}$$

where $R'(y) = 0$ for $y < 0$, $R'(y) = 1$ for $y > 0$, but $R'(0)$ is not defined. A problem shows up if $w_0 = 0$ because the function $x \mapsto R'(w_0 x)$ is not defined in this case. Of course, one can argue that the multiplication by $w_0 = 0$ is enough to obtain the correct solution $f' = 0$. However this operation is not correct on solid mathematical grounds because the function $x \mapsto R'(w_0 x)$ is not mathematically defined for $w_0 = 0$.

2.2 Second example

The second example is more problematic. It comes from Murat & Trombetti (2003) but is rewritten here with Neural Networks notation. Consider two functions f_0 and S_0. The function $f_0 : \mathbb{R} \to \mathbb{R}^2$ is linear

$$f_0(x) = (x, x) = W_0 x \text{ with } W_0 = (1, 1)$$

while the second function S_0 is a *maxpooling* function over two values

$$S_0(y) = \max(y_1, y_2), \text{ where } y = (y_1, y_2) \in \mathbb{R}^2. \tag{4}$$

Then $f = S_0 \circ f_0$ is the identity $f(x) = x$ for all $x \in \mathbb{R}$, so that $f' \equiv 1$ is of course bounded. The gradient of f_0 is $\nabla f_0 = W_0$. The gradient of S_0 is defined almost everywhere. There are three cases: the two first cases are

$$\text{if } y_1 > y_2 \text{ then } \nabla S_0(y) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ if } y_1 < y_2 \text{ then } \nabla S_0(y) = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

The third case is for $y_1 = y_2$, then $\nabla S_0(y)$ is not defined.

Since $f(x) = x$, the chain rule formula (2) writes

$$1 = \nabla S_0(f_0(x)) W_0. \tag{5}$$

Since $f_0(x) = (x, x)$ then $\nabla S_0(f_0(x))$ is nowhere defined. Since $W_0 \neq 0$, one can not argue as in the first example that the product with W_0 is enough to recover the correct solution. One obtains a paradox since the left hand side equal to 1 is perfectly known while the right hand side is not even a correctly defined function.

2.3 General case

The general case is that the chain rule formula (2) has no clear interpretation. In our opinion, it is related to some seemingly erratic behavior of nonsmooth autodiff related to the example Boustany (2024) detailed in Section 4.3.

3 The Murat-Trombetti Theorem

The Murat & Trombetti (2003) Theorem offers a natural solution to the apparent paradox explained in the previous examples. It is adapted in the sequel for the context of Neural Networks. We slightly adapt the notations from Murat & Trombetti (2003) and use two different notations for the gradient ∇f and for the associated gradient $\widetilde{\nabla} f$. The examples will show that associated gradients correspond to gradients calculated with autodiff.

We will need the notion of Lipschitz and piecewise C^1 functions $R^a \to \mathbb{R}^b$ which is defined as follows. We consider a finite decomposition of $\mathbb{R}^a$ in Lipschitz pieces P^α for finite number of values $1 \leq \alpha < \infty$

$$\mathbb{R}^a = \bigcup_\alpha P^\alpha, \qquad P^\alpha \cap P^\beta = \emptyset \text{ for } \alpha \neq \beta.$$

For the simplicity, the pieces P^α are piecewise affine. It means that they correspond to polygons, polyhedrons, lines, half lines, points and all finite unions and intersections of such objects in any dimension. This intuitive condition will be evident in our examples so we do not detail it and refer to Murat & Trombetti (2003). More general pieces are possible.

We assume that there exists functions $f^\alpha : \mathbb{R}^a \to \mathbb{R}^b$ with the regularity $f^\alpha \in \text{Lip}(\mathbb{R}^a : \mathbb{R}^b) \cap C^1(\mathbb{R}^a : \mathbb{R}^b)$ for all α such that

$$f(x) = \sum_\alpha \mathbf{1}_{P^\alpha}(x) f^\alpha(x) \quad \forall x \in \mathbb{R}^a \tag{6}$$

where the notation $\mathbf{1}_\omega$ denotes the indicatrix function of a set ω, that is $\mathbf{1}_\omega(x) = 1$ for $x \in \omega$ and $\mathbf{1}_\omega(x) = 0$ for $x \notin \omega$.

Definition 1. *The main idea in Murat & Trombetti (2003) is a gradient defined everywhere associated to the representation (6). We note it*

$$\widetilde{\nabla} f(x) = \sum_\alpha \mathbf{1}_{P^\alpha}(x) \nabla f^\alpha(x) \in \mathcal{M}_{b,a}(\mathbb{R}) \quad \forall x \in \mathbb{R}^a. \tag{7}$$

In brief we refer to it as an associated gradient.

The associated gradient is a real $b \times a$ matrix defined for all $x \in \mathbb{R}^a$. It is not unique since it depends on the representation (6) which is not unique. Corollary 1 will show that it is equal to the gradient almost everywhere.

Theorem 1 (Murat-Trombetti)**.** *Consider two functions $u \in Lip(\mathbb{R}^a : \mathbb{R}^b)$ and $v \in Lip(\mathbb{R}^b : \mathbb{R}^c)$. Assume v is piecewise C^1 so that it admits the representation (6) and has an associated gradient (7). Then the chain rule identity holds in $L^\infty(\mathbb{R}^a : \mathcal{M}_{c,a}(\mathbb{R}))$*

$$\nabla(v \circ u) = \widetilde{\nabla} v \circ u \ \nabla u.$$

Hint of the proof. The key part of the proof is the third step in (Murat & Trombetti, 2003, page 590) that we reproduce mutatis mutandi within the convenient functional setting. The two first steps in Murat & Trombetti (2003) are evident for u and v Lipschitz. The fourth and fifth steps concern additional properties.

• Since v is piecewise C^1, it admits a piecewise smooth approximations denote as $v^\alpha \in C^1(\mathbb{R}^b)$ for all α. Since $\nabla v^\alpha \circ u \in C^0(\mathbb{R}^a : \mathbb{R}^c)$ is a continuous function, the chain rule is applied without difficulty

$$\nabla(v^\alpha \circ u) = \nabla v^\alpha \circ u \ \nabla u. \tag{8}$$

This identity holds almost everywhere (a.e.) with respect to $x \in \mathbb{R}^b$. Let U^α be the measurable set

$$U^\alpha = \{x \in \mathbb{R}^a \ \ u(x) \in P^\alpha\}. \tag{9}$$

• One notes that

$$v(u(x)) = v^\alpha(u(x)) \quad a.e. \ \ x \in U^\alpha. \tag{10}$$

To use this identity one notes $w = v \circ u - v^\alpha \circ u$. Then one uses an intuitive but non trivial important property from Stampacchia (1963); Kinderlehrer & Stampacchia (2000) (a self contained proof is in the appendix). It writes

$$\nabla w(x) = 0 \quad a.e. \ \ x \in \{x \in \mathbb{R}^a : w(x) = 0\}. \tag{11}$$

Since $U^\alpha \subset \{x \in \mathbb{R}^a : w(x) = 0\}$, it yields using (8)

$$\nabla(v \circ u)(x) = \nabla v^\alpha \circ u(x) \ \nabla u(x) \quad a.e. \ \ x \in U^\alpha. \tag{12}$$

One also has

$$\mathbf{1}_{U^\alpha}(x) = \mathbf{1}_{P^\alpha}(u(x)) \quad a.e. \ \ x \in \mathbb{R}^a. \tag{13}$$

• Consider the difference of the two terms in the claim $A(x) = \nabla(v \circ u)(x) - \widetilde{\nabla} v \circ u \ \nabla u$. It writes also

$$\begin{array}{rcll}
A(x) & = & \nabla(v \circ u) - \left(\sum_\alpha \mathbf{1}_{P^\alpha}(u(x)) \nabla v^\alpha(u(x))\right) \nabla u(x) & \\
& = & \left(\sum_\alpha \mathbf{1}_{P^\alpha}(u(x))\right) \nabla(v \circ u)(x) - \left(\sum_\alpha \mathbf{1}_{P^\alpha}(u(x)) \nabla v^\alpha(u(x)) \nabla u(x)\right) & \\
& = & \sum_\alpha \mathbf{1}_{P^\alpha}(u(x)) \left(\nabla(v \circ u)(x) - \nabla v^\alpha(u(x)) \nabla u(x)\right) & \\
& = & \sum_\alpha \mathbf{1}_{U^\alpha}(x) \left(\nabla(v \circ u)(x) - \nabla v^\alpha(u(x)) \nabla u(x)\right) & \text{(use (13))} \\
& = & \sum_\alpha \mathbf{1}_{U^\alpha}(x) \, (0) & \text{(use (12))}
\end{array}$$

where all manipulations holds a.e. with respect to x. Therefore $A(x) = 0$ a.e. which is the claim. □

Corollary 1. *The associated gradient of Definition 1 is equal to the gradient ∇f almost everywhere with respect to x. That is $\widetilde{\nabla} f = \nabla f$ in the space $L^\infty(\mathbb{R}^a : \mathcal{M}_{a,a}(\mathbb{R}))$.*

Proof. Write Theorem 1 with $v = f$ and $u(x) = x$, so that $\nabla u = I$ is the identity matrix. $\square$

Corollary 2. *Consider a Neural Network function f defined by the composition formula (1) where all functions f_r and S_r are Lipschitz for $1 \le r \le \ell$. Assume moreover that all f_r and S_r are piecewise C^1 so that they admit associated gradients (7)*

$$\widetilde{A}_r = \widetilde{\nabla} f_r \circ S_r \circ \cdots \circ S_1 \circ f_0 \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_{r+1},a_0}(\mathbb{R}))$$

and

$$\widetilde{B}_r = \widetilde{\nabla} S_r \circ \cdots \circ S_1 \circ f_0 \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_r,a_0}(\mathbb{R})).$$

Then

$$\nabla f = \widetilde{A}_\ell(x)\widetilde{B}_\ell(x)\widetilde{A}_{\ell-1}(x)\widetilde{B}_{\ell-1}(x)\ldots\widetilde{A}_1(x)\widetilde{B}_1(x)\widetilde{A}_0(x) \quad a.e. \;\; x \in \mathbb{R}^a \tag{14}$$

where the right hand side is defined for all $x \in \mathbb{R}^a$.

Proof. The proof is based on iterations of Theorem 1.

- The first step is based on $f = f_\ell \circ (S_\ell \circ f_{\ell-1} \circ \ldots f_0)$. It yields

$$\nabla f = \widetilde{\nabla} f_\ell \circ S_\ell \circ f_{\ell-1} \circ \ldots\ldots f_0 \; \nabla S_\ell \circ f_{\ell-1} \circ \cdots \circ f_0.$$

The gradient ∇f is expressed as the product of one associated gradient and one gradient.

- The second step is based on $S_\ell \circ f_{\ell-1} \circ \ldots f_0 = S_\ell \circ (f_{\ell-1} \circ \ldots f_0)$. One obtains

$$\nabla S_\ell \circ f_{\ell-1} \circ \cdots \circ f_0 = \widetilde{\nabla} S_\ell \circ f_{\ell-1} \circ \ldots\ldots S_1 \circ f_0 \; \nabla f_{\ell-1} \circ \ldots S_1 \circ f_0$$

One substitutes this expression in the expression for ∇f which is now expressed as the product of two associated gradients and one gradient.

- Then iterations yields the result. The right hand side of the claim is defined for all x by definition of the associated gradients. $\square$

If the functions f_r are linear one can simplify using $\widetilde{A}_r(x) = W_r$ which is a constant matrix. One obtains the representation

$$\nabla f = W_\ell \widetilde{B}_\ell(x) W_{\ell-1} \widetilde{B}_{\ell-1}(x) \ldots W_1 \widetilde{B}_1(x) W_0 \quad a.e. \;\; x \in \mathbb{R}^a$$

where the right hand side is defined for all $x \in \mathbb{R}^a$.

4 Examples

We illustrate the interest of the Murat-Trombetti Theorem on simple examples where the number of layers is limited for the sake of simplicity.

4.1 Back to the first example

The issue is the value of the derivative of the ReLU function at the origin.

One can simply use three pieces $P^1 = (-\infty, 0)$, $P^2 = (0, \infty)$ and $P^3 = \{0\}$ to construct the associated derivative $\widetilde{R}'$. The three smooth functions are f^1, f^2 and f^3. By definition $f^1(x) = R(x) = 0$ for $x \in P^1$, then $(f^1)'(x) = 0$ for $x \in P^1$. Similarly $f^2(x) = R(x) = x$ for $x \in P^2$, then $(f^2)'(x) = 1$ for $x \in P^2$. The only constraint on f^3 is $f^3(0) = 0$, so $(f^3)'(0)$ can be any real number. Let us note $(f^3)'(0) = z \in \mathbb{R}$.

So the associated derivative is

$$\widetilde{R}'(x) = 0 \text{ for } x < 0, \quad \widetilde{R}'(x) = 1 \text{ for } x > 0, \quad \widetilde{R}'(x) = z \text{ for } x = 0.$$

Clearly the associated derivative depends on the choice of z, so a better but heavier notation would have been $\widetilde{R}'_z$ for the associated derivative. Most of the studies about the influence of the derivative of the ReLU at the origin are restricted to $z = 0$ that is to $\widetilde{R}'(0) = 0$, see Boustany (2024); Berner et al. (2019); Bertoin et al. (2021). The previous detailed analysis shows that $z \neq 0$ is also possible.

Whatever the value of z, then (3) becomes $f' = \widetilde{R}' w_0$ which is non ambiguous for $w_0 = 0$ since $\widetilde{R}'$ is correctly defined. However, once again, the use of an associated gradient is not mandatory in this case since there is no real difficulty for $w_0 = 0$.

4.2 Back to the second example

To construct an associated gradient for the *maxpooling* function S_0 (4), one can distinguish three pieces which are $P^1 = \{(y_1, y_2) \in \mathbb{R}^2 \ : \ y_1 < y_2\}$, $P^2 = \{(y_1, y_2) \in \mathbb{R}^2 \ : \ y_1 > y_2\}$ and $P^3 = \{(y_1, y_2) \in \mathbb{R}^2 \ : \ y_1 = y_2\}$. Then the smooth functions are f^1, f^2 and f^3. Clearly $f^1(y_1, y_2) = y_1$ in P^1, so that $\nabla f^1(y_1, y_2) = (1, 0)$ in P^1. For similar reasons, $\nabla f^2(y_1, y_2) = (0, 1)$ in P^2.

The critical situation concerns P^3. The construction principle (6) yields that $f^3(y_1, y_2) = y_1 = y_2$ in P^3. It can be written $f^3(y, y) = y$ for all $y \in \mathbb{R}$. The function f^3 being continuously differentiable, one has necessarily $\partial_{y_1} f^3(y, y) + \partial_{y_2} f^3(y, y) = 1$ for all y, that is

$$\partial_{y_1} f^3(y_1, y_2) + \partial_{y_2} f^3(y_1, y_2) = 1 \text{ on } P^3. \tag{15}$$

Let us now examine what is the meaning of the modified chain rule formula which replaces the initial one (5). This modified chain rule formula can be taken from Corollary 2

$$1 = \widetilde{\nabla} S_0(f_0(x)) W_0 \text{ for all } x \in \mathbb{R}. \tag{16}$$

The key observations are that $W_0 = (1, 1)$ and that $\widetilde{\nabla} S_0(f_0(x)) = \nabla f^3(f_0(x))$ since $f_0(x) \in P^3$. Then (16) reduces to the identity (15) which holds by definition. So the paradox does not show up again.

Remark 1. *A simple geometrical interpretation emerges from the fact that (16) reduces to (15). Actually the gradient of f^3 can take any value in the direction* **normal** *to the line P^3 while it takes the correct value in the direction* **tangent** *to P^3.*

4.3 The Boustany example

This example is proposed in Boustany (2024) to exemplify the issues at stake with nonsmooth autodiff with *maxpoooling* functions. The example is directly implemented in PyTorch. One defines a first maximum function $\max_1$ for a vector $x \in \mathbb{R}^a$ of arbitrary size $a \geq 1$. The scripts taken from Boustany (2024) are in Table 1, together with a second maximum function $\max_2$ which is a PyTorch function. Then for given $x \in \mathbb{R}^a$, one defines the function

$$t \mapsto f(t) = \max_1(tx) - \max_2(tx).$$

```
def max_1(x):
res = x[0]
for i in range(1, a):
if x[i] > res:  res = x[i]
return res
```

```
def max_2(x):
return torch.max(x)
```

Table 1: Script of the functions $\max_1$ and $\max_2$

By construction f is the null function.

However it is reported in (Boustany, 2024, Table 1) that the derivative calculated with *autodiff* in PyTorch is not zero. More precisely take $x = (1, 2, 3, 4)$, then $f'(t)$ is (numerically) zero everywhere except at $t = 0$ where the derivative is ≈ -1.5. Our own tests reported in Table 2 confirm this observation.

The explanation in the context of the Murat-Trombetti representation formula (7) is as follows. The function $\max_1$ calculated with *autodiff* is given in Table 1. It yields that the associated gradient of $\max_1$ is calculated

t	-1	-0.5	-0.01	0	0.01	0.5	1
derivative of f	0	0	0	-1.5	0	0	0

Table 2: Values of the derivative of f calculated with autodiff within PyTorch

accordingly to the representation

$$\begin{array}{ll} P^1 = \{x \in \mathbb{R}^4 |\ x_1 \geq \max(x_2, x_3, x_4)\}, & f^1(x) = x_1, \\ P^2 = \{x \in \mathbb{R}^4 |\ x_1 < x_2 \text{ and } x_2 \geq \max(x_3, x_4)\}, & f^2(x) = x_2, \\ P^3 = \{x \in \mathbb{R}^4 |\ \max(x_1, x_2) < x_3 \text{ and } x_3 \geq x_4\}, & f^3(x) = x_3, \\ P^4 = \{x \in \mathbb{R}^4 |\ \max(x_1, x_2, x_3) < x_4\}, & f^4(x) = x_4 \end{array} \tag{17}$$

where $x = (x_1, x_2, x_3, x_4)$. Then the associated gradient is

$$\widetilde{\nabla}\max{}_1(x) = \quad (1, 0, 0, 0) \text{ in } P^1, \quad (0, 1, 0, 0) \text{ in } P^2, \quad (0, 0, 1, 0) \text{ in } P^3, \quad (0, 0, 0, 1) \text{ in } P^4. \tag{18}$$

The representation that we propose for $\max_2$ is different. It is based on

$$\begin{array}{ll} P^1 = \{x \in \mathbb{R}^4 |\ x_1 > \max(x_2, x_3, x_4)\}, & f^1(x) = x_1, \\ P^2 = \{x \in \mathbb{R}^4 |\ x_2 > \max(x_1, x_3, x_4)\}, & f^2(x) = x_2, \\ P^3 = \{x \in \mathbb{R}^4 |\ x_3 > \max(x_1, x_2, x_4)\}, & f^3(x) = x_3, \\ P^4 = \{x \in \mathbb{R}^4 |\ x_4 > \max(x_1, x_2, x_3)\}, & f^4(x) = x_4, \\ P^5 = \{x \in \mathbb{R}^4 |\ x_1 = x_2 > \max(x_3, x_4)\}, & f^5(x) = (x_1 + x_2)/2, \\ P^6 = \{x \in \mathbb{R}^4 |\ x_1 = x_3 > \max(x_2, x_4)\}, & f^6(x) = (x_1 + x_3)/2, \\ P^7 = \{x \in \mathbb{R}^4 |\ x_1 = x_4 > \max(x_2, x_3)\}, & f^7(x) = (x_1 + x_4)/2, \\ P^8 = \{x \in \mathbb{R}^4 |\ x_2 = x_3 > \max(x_1, x_4)\}, & f^8(x) = (x_2 + x_3)/2, \\ P^9 = \{x \in \mathbb{R}^4 |\ x_2 = x_4 > \max(x_1, x_3)\}, & f^9(x) = (x_2 + x_4)/2, \\ P^{10} = \{x \in \mathbb{R}^4 |\ x_3 = x_4 > \max(x_1, x_2)\}, & f^{10}(x) = (x_3 + x_4)/2, \\ P^{11} = \{x \in \mathbb{R}^4 |\ x_1 = x_2 = x_3 > x_4\}, & f^{11}(x) = (x_1 + x_2 + x_3)/3, \\ P^{12} = \{x \in \mathbb{R}^4 |\ x_1 = x_2 = x_4 > x_3\}, & f^{12}(x) = (x_1 + x_2 + x_4)/3, \\ P^{13} = \{x \in \mathbb{R}^4 |\ x_1 = x_3 = x_4 > x_2\}, & f^{13}(x) = (x_1 + x_3 + x_4)/3, \\ P^{14} = \{x \in \mathbb{R}^4 |\ x_2 = x_3 = x_4 > x_1\}, & f^{14}(x) = (x_2 + x_3 + x_4)/3, \\ P^{15} = \{x \in \mathbb{R}^4 |\ x_1 = x_2 = x_3 = x_4\}, & f^{15}(x) = (x_1 + x_2 + x_3 + x_4)/4. \end{array} \tag{19}$$

It yields

$$\widetilde{\nabla}\max{}_2(x_1, x_2, x_3, x_4) = \frac{1}{N(x_1, x_2, x_3, x_4)} (y_1, y_2, y_3, y_4) \tag{20}$$

where

- $N(x_1, x_2, x_3, x_4)$ is the number of values x_i $(1 \leq i \leq 4)$ equal to the maximum value $\max(x_1, x_2, x_3, x_4)$,
- $y_i = 1$ if $x_i = \max(x_1, x_2, x_3, x_4)$, and $x_i = 0$ if $a_i < \max(x_1, x_2, x_3, x_4)$.

Remark 2. *The examination of (19) shows that $\widetilde{\nabla}\max_2$ is symmetrized. One has for example*

$$\widetilde{\nabla}\max{}_2(\mathbf{0}) = \frac{1}{4}(1, 1, 1, 1), \qquad \mathbf{0} = (0, 0, 0, 0). \tag{21}$$

More precisely, the associated gradient $\widetilde{\nabla}\max_2$ is invariant under the action of permutations among the values equal to the maximum. This is confirmed by numerical evidence based on elementary tests in PyTorch.

Now let us calculate the associated derivative of f at $t = 0$ with the rules of automatic differentiation

$$\widetilde{f}'(0) = \frac{d}{dt}\max{}_1(t, 2t, 3t, 4t)_{|t=0} - \frac{d}{dt}\max{}_2(t, 2t, 3t, 4t)_{|t=0}$$

that is

$$\widetilde{f}'(0) = \widetilde{\nabla}\max{}_1(\mathbf{0}) \cdot (1, 2, 3, 4) - \widetilde{\nabla}\max{}_2(\mathbf{0}) \cdot (1, 2, 3, 4).$$

One obtains $\widetilde{f}'(0) = (1, 0, 0, 0) \cdot (1, 2, 3, 4) - \frac{1}{4}(1, 1, 1, 1) \cdot (1, 2, 3, 4) = 1 - 10/4 = -1.5$ which is the value reported in Boustany (2024) and in Table 2.

4.4 A simplified Boustany example

Finally we prepare another simple example in the spirit of Boustany (2024), but where the maximum is calculated with the *MaxPool1d* function of PyTorch. Maxpooling is an important and popular operation in modern neural networks.

We assemble the function $f(t) = \text{maxpool1d}(t, 4t) - \text{maxpool1d}(4t, t)$. Some values implemented in PyTorch are given in Table 3. The explanation of the value of $f'(0)$ is as follows. Actually all numerical test show that the PyTorch function maxpool1d = torch.nn.MaxPool1d$(2, \mathit{stride} = 1)$ with a window of 2 elements has the same associated gradient as the function $\max_1$ presented in Table 1. Therefore $\widetilde{\nabla}\text{maxpool1d}(0, 0) = (1, 0)$. Then $\widetilde{f}'(0) = (1, 0) \cdot (1, 4) - (4, 1)(1, 0) \cdot (1, 4) = -3$ which is the value in Table 3 observed in the numerical tests.

t	-1	-0.5	-0.01	0	0.01	0.5	1
$f'(t)$	0	0	0	-3	0	0	0

Table 3: Values of the derivative calculated with autodiff within PyTorch

5 Conclusion

The Murat-Trombetti Theorem provides a simple functional framework which allows to manipulate composition of Lipschitz and piecewise C^1 functions. It constructs an associated gradient which is defined for all values of the input variable. An associated gradient is not unique nevertheless. We have observed that the gradient obtained from nonsmooth autodiff in PyTorch is systematically equal to an associated gradient in the sense of Murat-Trombetti. This approach also provides a non ambiguous chain rule formula, that was actually the key in the original paper Murat & Trombetti (2003).

Connections with the method of breakpoints described in Daubechies et al. (2019) is a priori possible. We mention some problems which could be the subject for future researches. Some of them are already evoked in Berner et al. (2019).

Evaluation of the Lipschitz constant of a function modeled with Neural Network: The numerical evaluation and use of the Lipschitz constant of a given Neural Network function where the weights W_r and the biaises b_r are given has the subject of recent researches Combettes & Pesquet (2020); Virmaux & Scaman (2018); Pintore & Després (2024); Béthune (2024). More solid foundations for these works can be obtained with associated gradients.

More variables and training: The associated gradient has been introduced and studied in this work on examples with limited number of layers and with limited number of variables. In practice a Neural Network function is defined with respect to space variables (typically x in (1)) and to parameters (typically W_r and b_r in (1)). Then it raises the mathematical question of the definition of an associated gradient with respect to all variables. There is major practical interest in designing an associated gradient with respect to the parameters W_r and b_r only. It can be used to describe a functional setting for training sequences and to compare with the numerical tests in Boustany (2024).

References

Luigi Ambrosio and Gianni Dal Maso. A general chain rule for distributional derivatives. *Proceedings of the American Mathematical Society*, 108(3):691–702, 1990.

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

Julius Berner, Dennis Elbrächter, Philipp Grohs, and Arnulf Jentzen. Towards a regularity theory for relu networks–chain rule and global error estimates. In *2019 13th International conference on Sampling Theory and Applications (SampTA)*, pp. 1–5. IEEE, 2019.

David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. Numerical influence of relu'(0) on backpropagation. *Advances in Neural Information Processing Systems*, 34:468–479, 2021.

David Bertoin, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. Erratum: Numerical influence of relu'(0) on backpropagation. Technical report, 2023. URL `https://hal.science/hal-03265059/file/Impact_of_ReLU_prime.pdf`.

Louis Béthune. *Deep learning with Lipschitz constraints.* PhD thesis, Université de Toulouse, 2024.

Jérôme Bolte and Edouard Pauwels. Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning. *Mathematical Programming*, 188:19–51, 2021.

Ryan Boustany. On the numerical reliability of nonsmooth autodiff: a maxpool case study. *Transactions on Machine Learning Research*, 2024. URL `https://arxiv.org/abs/2401.02736`.

Frank H Clarke. *Optimization and nonsmooth analysis.* SIAM, 1990.

Patrick L Combettes and Jean-Christophe Pesquet. Lipschitz certificates for layered network structures driven by averaged activation operators. *SIAM J. on Math. of Data Science*, 2(2):529–557, 2020.

Ingrid Daubechies, Ronald A. DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu neural networks. *Constructive Approximation*, 55:127–172, 2019.

Lawrence Craig Evans. *Measure theory and fine properties of functions.* Routledge, 2018.

David Kinderlehrer and Guido Stampacchia. *An introduction to variational inequalities and their applications.* SIAM, 2000.

Charles Bradfield Morrey Jr. *Multiple integrals in the calculus of variations.* Springer Science & Business Media, 2009.

François Murat and Cristina Trombetti. A chain rule formula for the composition of a vector-valued function by a piecewise smooth function. *Bollettino dell'Unione Matematica Italiana*, 6(3):581–595, 2003.

Moreno Pintore and Bruno Després. Computable lipschitz bounds for deep neural networks. Technical report, 2024. URL `https://inria.hal.science/hal-04756410`.

Hans Rademacher. Über partielle und totale differenzierbarkeit von funktionen mehrerer variabeln und über die transformation der doppelintegrale. *Mathematische Annalen*, 79(4):340–359, 1919.

Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Sci*, 6(12):310–316, 2017.

Guido Stampacchia. Equations elliptiques du second ordre à coefficients discontinus. *Séminaire Jean Leray*, (3):1–77, 1963.

Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

A A self contained proof of the Stampacchia property

The Stampacchia property Stampacchia (1963); Kinderlehrer & Stampacchia (2000) states that a function $w \in \mathrm{Lip}(R^a)$ is such that

$$\nabla w(x) = 0 \quad a.e. \;\; x \in \{x \in \mathbb{R}^a \; : \; w(x) = 0\}. \tag{22}$$

A simple proof comes from a regularization technique. See also Evans (2018).
First regularization. Consider $x \mapsto |x|_\varepsilon = \sqrt{x^2 + \varepsilon}$ for $\varepsilon > 0$. The derivative is $\frac{d}{dx}|x|_\varepsilon = \frac{x}{\sqrt{x^2+\varepsilon}}$.

Thanks to the Rademacher Theorem, w admits a gradient $\nabla w \in L^\infty(R^a)^a$. Also $|w|$ admits a gradient $\nabla |w| \in L^\infty(R^a)^a$ as well because $|w|$ is also Lipschitz. Take a vectorial smooth test function with compact support $\varphi \in C_0^1(\mathbb{R}^a)$. The integration by part formula holds

$$\int \nabla |w|(x)\cdot \varphi(x)dx = -\int |w|(x)\nabla\cdot\varphi(x)dx = -\lim_{\varepsilon\to 0^+}\int |w|_\varepsilon(x)\nabla\cdot\varphi(x)dx.$$

There is no difficulty in passing to the limit because w is continuous. A reverse integration by parts shows that

$$\begin{aligned}&-\textstyle\int |w|_\varepsilon(x)\nabla\cdot\varphi(x)dx = \int \nabla|w|_\varepsilon(x)\cdot\varphi(x)dx = \int \frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx\\ &= \textstyle\int_{w(x)>0}\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx + \int_{w(x)<0}\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx + \int_{w(x)=0}\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx.\end{aligned}$$

In the right hand side, the last integral vanishes of course. In the first integral one has the boundedness $\left|\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)\right| \le |\nabla w(x)\cdot\varphi(x)|$ where on the right hand side the function $x\mapsto|\nabla w(x)\cdot\varphi(x)|$ defines a function in $L^1(\mathbb{R}^a)$. One also has pointwise convergence almost everywhere with respect to x $\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x) \to \mathrm{sign}(w(x))\nabla w(x)\cdot\varphi(x)$ a.e. x. Therefore the Lebesgue dominated convergence Theorem yields

$$\lim_{\varepsilon\to 0^+}\int_{w(x)>0}\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx = \int_{w(x)>0}\nabla w(x)\cdot\varphi(x)dx.$$

Similarly $\lim_{\varepsilon\to0^+}\int_{w(x)<0}\frac{w(x)}{\sqrt{w(x)^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx = -\int_{w(x)<0}\nabla w(x)\cdot\varphi(x)dx$. It yields the formula

$$\int \nabla|w|(x)\cdot\varphi(x)dx = \int_{w(x)>0}\nabla w(x)\cdot\varphi(x)dx - \int_{w(x)<0}\nabla w(x)\cdot\varphi(x)dx. \tag{23}$$

Second regularization. Let us now redo the calculation but starting from a different regularization of the absolute value. We take $x\mapsto |x|^\varepsilon = \sqrt{(x+\sqrt{\varepsilon})^2+\varepsilon}$ for $\varepsilon>0$, with derivative $\frac{d}{dx}|x|^\varepsilon = \frac{x+\sqrt{\varepsilon}}{\sqrt{(x+\sqrt{\varepsilon})^2+\varepsilon}}$.

One has $\int\nabla|w|(x)\cdot\varphi(x)dx = -\int|w|(x)\nabla\cdot\varphi(x)dx = -\lim_{\varepsilon\to0^+}\int|w|^\varepsilon(x)\nabla\cdot\varphi(x)dx$ and

$$\begin{aligned}&-\textstyle\int|w|^\varepsilon(x)\nabla\cdot\varphi(x)dx = \int\nabla|w|^\varepsilon(x)\cdot\varphi(x)dx = \int\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx\\ &= \textstyle\int_{w(x)>0}\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx + \int_{w(x)<0}\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx\\ &+\textstyle\int_{w(x)=0}\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx\\ &= \textstyle\int_{w(x)>0}\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx + \int_{w(x)<0}\frac{w(x)+\sqrt{\varepsilon}}{\sqrt{(w(x)+\sqrt{\varepsilon})^2+\varepsilon}}\nabla w(x)\cdot\varphi(x)dx\\ &+\textstyle\frac{1}{\sqrt{2}}\int_{w(x)=0}\nabla w(x)\cdot\varphi(x)dx\end{aligned}$$

The Lebesgue dominated convergence Theorem yields the same limit as before for the two first integrals. However the third integral remains. One obtains

$$\int\nabla|w|(x)\cdot\varphi(x)dx = \int_{w(x)>0}\nabla w(x)\cdot\varphi(x)dx - \int_{w(x)<0}\nabla w(x)\cdot\varphi(x)dx + \frac{1}{\sqrt{2}}\int_{w(x)=0}\nabla w(x)\cdot\varphi(x)dx. \tag{24}$$

Final part of the proof. Comparison of (23) and (24) yields $\int_{w(x)=0}\nabla w(x)\cdot\varphi(x)dx = 0$ for all $\varphi\in C_0^1(\mathbb{R}^a)$. This is equivalent to the Stampacchia property (22) because the test function φ is arbitrary.